

# A Hierarchical Multi-Output Nearest Neighbor Model for Multi-Output Dependence Learning

Richard G. Morris, Tony Martinez, Michael R. Smith

Brigham Young University, Provo, UT, 84602, USA,  
 rmmorris@axon.cs.byu.edu, martinez@cs.byu.edu, msmith@axon.cs.byu.edu

**Abstract.** Multi-Output Dependence (MOD) learning is a generalization of standard classification problems that allows for multiple outputs that are dependent on each other. A primary issue that arises in the context of MOD learning is that for any given input pattern there can be multiple correct output patterns. This changes the learning task from function approximation to relation approximation. Previous algorithms do not consider this problem, and thus cannot be readily applied to MOD problems. To perform MOD learning, we introduce the Hierarchical Multi-Output Nearest Neighbor model (HMONN) that employs a basic learning model for each output and a modified nearest neighbor approach to refine the initial results.

## 1 Introduction

In this paper, we introduce Multi-Output Dependence (MOD) learning as an algorithmic family that models dependencies between multiple outputs. Traditional supervised learning seeks to map an input vector  $\mathbf{x}$  to an output vector  $\mathbf{y} \in C$  where  $C$  is set of possible outputs. Further, multi-label classification specifically examines the case where multiple target labels must be assigned to each instance [6] while structured prediction predict multiple target labels where  $\mathbf{y}$  is structured. MOD addresses problems where the outputs are dependent on each other and where there are multiple correct output vectors  $\mathbf{y}$  for a given  $\mathbf{x}$ . Any one output may be considered correct or incorrect only when considered in the context of other outputs.

An example MOD problem is the following. Assume we want to propose an action for a company to take that generates sales and/or retains a customer. For example, a particular customer may be contemplating switching to a competitor. What should the company do to retain this customer? There could be multiple correct actions. A sales person could write the customer and offer incentives for staying, or the CEO could call the customer to express how important he is to them. Of course, each action incurs a certain cost. Having the CEO call a customer is more expensive than having a help-desk employee write the customer an e-mail, but both are viable solutions. However, if that customer happens to be the largest source of revenue for that company, then sending a generic e-mail may not be the best course of action to take. It may be the case where calling

would only be correct *if* the CEO made the phone call but writing an e-mail would be correct *if* it came from a different person.

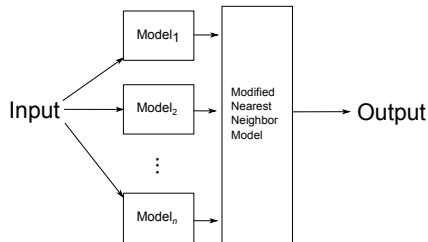
MOD problems can be seen as those problems where the outputs are important in addition to the inputs when making a decision. MOD learning requires approximating a relation, as opposed to the more traditional function approximation. We define a training data set  $T$  to be a set of input vectors  $\mathbf{x}$  each labeled with an appropriate output  $\mathbf{y}$ . An input vector  $\mathbf{x}$  can be associated with multiple output vectors  $\mathbf{y}$  where  $|\mathbf{y}| \geq 1$ . In this case, there are multiple correct outputs for  $\mathbf{x}$ . Some outputs may be more desirable than others, but there are still multiple outputs that would be acceptable given the input  $\mathbf{x}$ . This changes the task from finding a mapping function  $\mathbf{x} \mapsto \mathbf{y}$  to finding a relation from  $\mathbf{x}$  to  $\mathbf{y}$ . We consider the relation where there are multiple outputs (where  $|\mathbf{y}| > 1$ ) and there is a dependency between the outputs. This gives rise to interesting questions about which correct solutions to choose when there are multiple correct solutions available.

Many current algorithms fail to directly support multiple outputs. Current approaches either induce one model per output or create a single model that gives multiple outputs without explicitly modeling the dependencies. Different models support multiple independent outputs with a varying degree of success, without further modification. Decision Tree learning algorithms must either induce multiple trees in order to produce multiple outputs or must induce a single tree that blows up exponentially, but neither of these approaches can model dependence between the output variables.  $K$ -Nearest Neighbor algorithms can support multiple outputs with little change to the basic algorithm. Multi-Layer Perceptron (MLP) models can give multiple outputs with a single model or multiple model approach. However, none of these algorithms explicitly model dependent outputs. Auto-associative models, such as Hopfield networks [7], come close to this capability, but they are unable to handle arbitrary input and output mappings in contrast to the hetero-associative model that we present.

We introduce the Hierarchical Multi-Output Nearest Neighbor model (HMONN) in order to solve the MOD problem. This hierarchical model has two layers. The first layer is a naïve approach with one learning model per output. The models that comprise the first layer can be any traditional machine learning model. The second layer is a modified nearest neighbor model that refines the predictions made on the first layer. HMONN is shown graphically in Figure 1. Though HMONN focuses on tasks with nominal features, it also gives improvement for some tasks with real-valued features by implicitly modeling a similarity function for the feature space.

## 2 Related Work

Other work has examined classification with multiple labels, although the labels are generally not considered to be dependent on each other and multiple correct labels are not considered. Multi-label classification considers problems with multiple outputs, but no dependency between the outputs is modeled. Tsoumakas



**Fig. 1.** The HMONN model used to solve MOD problems. The models used as input to the modified nearest neighbor model can use any algorithm to produce an initial prediction. This models the dependence between the outputs in terms of the local context from the nearest neighbor algorithm.

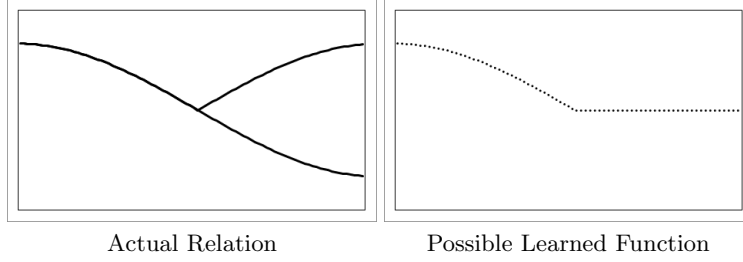
et al. [12] give an overview of multi-label classification. They define two main approaches for multi-label classification. The first approach is problem transformation, where the given data is transformed into a single problem that already has a well defined solution. The second approach is algorithm adaptation, where current algorithms are modified to solve the multi-label classification problem. Recent work has looked at correlations between labels in multi-label classification to improve accuracy [5]. Read [9,10] introduced chain classifiers for supporting these correlations which could be viable for supporting MOD problems.

Many problems have a structure that is missed by standard classification algorithms [11]. Structured Prediction (SP) seeks to solve this problem by modeling the structure of the outputs. This structure could be a sequence, a tree, a graph, or an image. This allows for multi-output as well as output dependencies. However, these dependencies are almost always limited to Markovian dependencies — related by time or space. Theoretically, SP algorithms are capable of modeling any problem with structure, and MOD problems would be an example of this kind of problem. The main difference between MOD and SP is that MOD problems are assumed to have some inputs with multiple correct outputs, whereas with current SP algorithms there is a single correct output assumed for each input. Bakir et al. [1] give an overview of the state of the art in SP.

While MOD learning is relation approximation, this should not be confused with relational learning. Statistical Relational Learning [8,4] and Multi-Relational Learning [2] both handle relational data, not relation approximation. These relational learning models learn a function from relational data and handle specially formatted and structured data.

### 3 HMONN

We present the *Hierarchical Multi-Output Nearest Neighbor* model (HMONN) to solve the MOD problem. We define an output prediction  $\hat{\mathbf{y}}$  to be correct for a given input vector  $\mathbf{x}$  if there is some training instance in the training data  $T$  that has  $\mathbf{x}$  labeled with output  $\mathbf{y}$  and  $\hat{\mathbf{y}} = \mathbf{y}$  (or if  $\hat{\mathbf{y}} = \mathbf{y}$  for the current



**Fig. 2.** A graphical example of a relation with multiple correct outputs. The solid curve represents the relation itself. The dotted curve represents the function learned by an MLP model. The dotted curve follows the solid curve exactly until the relation branches, at which point the dotted curve is in the center of the two branches.

test instance). The traditional definition of a correct prediction only takes into account the labeling on the instance currently being tested. This definition allows the model to use information contained within the training data to determine which output predictions should be counted as correct.

Traditional models are not able to model the dependencies between outputs. This is, in part, due to the fact that traditional models are function approximators, and MOD problems are relational. As an example of this, consider a training set that contains two training patterns with the same  $\mathbf{x}$  and different  $\mathbf{y}$ . A traditional MLP will oscillate between the multiple possible outputs, and may not give any of the possible correct output vectors. An MLP will adjust weights towards outputting  $\{1, 0\}$  whenever the first instance is encountered, and whenever the second instance is encountered it will adjust the weights towards outputting  $\{0, 1\}$ . The network will consequently adjust the weights towards the output  $\{0.5, 0.5\}$  (without ever stabilizing), rather than towards either of the correct outputs. A graphical example of the problem faced by an algorithm trying to learn a problem with multiple correct outputs is shown in Figure 2. The solid curve represents the relation in the training data and the dotted curve represents the function that could be learned, for example by an MLP. An appropriate algorithm should output both branches of the relation, following the relation exactly, choose one of the branches arbitrarily, or choose one based on some criteria. It should not, however, output something completely different.

HMONN favors one output vector over the others. Even though we could give a distribution of potential outputs from the neighborhood of the initial prediction, this version gives one of the possible correct output vectors for the given input vector  $\mathbf{x}$ . This output vector is the most common among the given neighborhood, and thus varies with neighborhood size and makeup. HMONN is a first step towards solving the MOD problem. HMONN starts with an initial prediction and then uses the extra information provided by that initial prediction to give the output. The initial prediction is obtained using any machine learning method. Here, we choose to train an MLP classifier for each output. The outputs from each MLP classifier are combined into an initial prediction. We present

a modified  $K$ -Nearest Neighbor (KNN) algorithm to give the final prediction. HMONN uses a different distance function where the initial prediction is used as part of the features for the distance function:

$$Dist(\{\mathbf{x}_1, \mathbf{y}_1\}, \{\mathbf{x}_2, \mathbf{y}_2\}) = \sqrt{\theta \sum_{i=1}^N (x_{1,i} - x_{2,i})^2 + (1 - \theta) \sum_{i=1}^M (y_{1,i} - y_{2,i})^2} \quad (1)$$

where  $N$  is the number of features in the input space,  $M$  is the number of outputs, and  $\theta$  is a weight on the range  $[0, 1]$ . The value for  $\theta$  emphasizes either the input space or the output space as more important. This modification of KNN captures the dependency between output variables by incorporating them into the input feature space. HMONN takes the initial prediction from the MLP classifiers, uses this initial prediction as part of the features in a KNN algorithm, and chooses the majority output vector from the neighborhood as the final prediction.

The relationship between the dependence between outputs and multiple correct output vectors for a given input vector is shown in Theorem 1. Theorem 1 claims that we can observe the dependence between two output variables directly in the training data. There is also a case for loose dependence that relies on different  $\mathbf{x}$  vectors being only similar, but this work considers exact equality.

**Theorem 1.** *Given random variables  $\mathbf{x}$ ,  $y_i$ , and  $y_j$ , where  $\mathbf{x}$  is an input vector of nominal features and  $y_i$  and  $y_j$  are scalars from the output vector  $\mathbf{y}$ , if the two output variables,  $y_i$  and  $y_j$ , are conditionally dependent on each other given the input  $\mathbf{x}$  and the training data  $T$ , then there is some input vector,  $\mathbf{x}$ , associated with multiple output vectors,  $\mathbf{y}$ , in  $T$ .*

*Proof.* Assume that outputs  $y_1$  and  $y_2$  are conditionally dependent given the input variable  $X$  and the training data  $T$ . By the definition of statistical dependence this implies that, for some input vector  $\mathbf{x}$ ,  $P(y_1 | y_2, \mathbf{x}, T) \neq P(y_1 | \mathbf{x}, T)$ . Assume that the output vector  $\mathbf{y} = [y_1, y_2]^T$  is the only possible correct output for  $\mathbf{x}$ . Then it is the case that  $P(y_1 | y_2, \mathbf{x}, T) = P(y_1 | \mathbf{x}, T) = 1$ . This contradicts the definition of statistical dependence. Thus, there must be multiple possible output vectors for the input vector  $\mathbf{x}$ .  $\square$

## 4 Experimental Results

The accuracy of MOD classifiers was evaluated on three different types of data: synthetic data, UCI repository data, and real-world data. This accuracy was compared to a baseline model that consists of a single classifier trained separately for each output, which we call the naïve model, where each separate prediction is combined into a single output vector. Accuracy is defined as follows.

$$MOD\_accuracy = \frac{\sum_i^D I(\{\mathbf{x}_i, \mathbf{z}_i\} \in T \cup \{D_i\})}{|D|} \quad (2)$$

where  $D$  is the test set,  $\mathbf{z}_i$  is the predicted output vector for instance  $\mathbf{x}_i$ ,  $T$  is the training set, and  $I(x)$  is the Kronecker delta function returning 1 if the

expression  $x$  is true and 0 otherwise. This accuracy metric counts a prediction as correct if an input vector  $\mathbf{x}$  in the data set is labeled with the predicted output vector  $\mathbf{z}$ . This considers all correct output vectors as equally good.

Standard machine learning tasks with only nominal input features are common, and we assume that the same will hold for MOD data sets. HMONN shows clear improvement on these data sets. Many tasks also have real-valued features. While it is more difficult to find a duplicate  $\mathbf{x}$  in these data sets, real-valued features will often have some level of discretization done to them, through either binning or rounding that increases the likelihood of finding duplicate  $\mathbf{x}$  vectors in the data set. This alters the amount of dependence between the output variables. Thus, in many current data sets, real-valued features do not necessarily take on a large range of values. This allows the given definition of accuracy to work in many cases with real-valued features. To better handle real-valued features, the definition of accuracy could be extended to allow for *similar* values, as opposed to requiring values to be exactly equal. We are currently working on extending MOD accuracy metrics to better support real-valued features.

Despite the issue of the frequency of exact  $\mathbf{x}$  vectors for real-valued features, HMONN improves the accuracy in some of the experiments on synthetic and UCI data that have real-valued inputs. This is due to the fact that the nearest neighbor portion of the algorithm creates an implicit similarity function for the feature space. The similarity function behaves differently based on the neighborhood size. This gives a distance-based voting for which outputs are correct for any given portion of the feature space. This causes the majority class for any given neighborhood in the feature space to always be the correct value. Selecting outputs in this fashion avoids some of the difficulty with real-valued features, even though it does not solve the problem completely. We are currently exploring ways to fully resolve this problem as future work.

Some initial experimentation was used to determine values for  $k$  and  $\theta$ . We tested values of  $k$  from 1 to 11 and values of  $\theta$  from  $\{0, 0.25, 0.5, 0.75, 1\}$ . We found that there was little difference between values of  $k$  and  $\theta$  except for  $\theta = 0$ , which performed slightly worse. In the following experiments, we use representative values  $k = 7$ , allowing for a reasonably sized neighborhood, and  $\theta = 0.5$ , to give an equal balance between the input and output features. Experiments are run using 10-fold cross validation. The naïve neural network layer had a standard MLP with a single hidden layer of  $2n$  nodes for each output with  $n$  being the number of attributes, including the outputs, in the corresponding data set. All experiments are run with a learning rate of 0.1 and stop after 10 epochs without any improvement on a held-out validation set. Statistical significance is determined using the Wilcoxon signed rank test with significance at  $p < 0.05$ .

#### 4.1 Synthetic Data

Two different types of synthetic data were created. One used real-valued features in order to determine whether HMONN implicitly models a similarity function for the feature space, as hypothesized. The other used only nominal features.

**Table 1.** Results comparing HMONN to the naïve model for real-valued features. Bold values indicate that the values are statistically significant. The  $p$ -value for the total is  $p < .0001$ .

	Real-Valued Features				Nominal Features			
Model	2-Output	3-Output	4-Output	Total	2-Output	3-Output	4-Output	Total
HMONN	<b>0.760</b>	<b>0.877</b>	<b>0.905</b>	<b>0.847</b>	<b>0.703</b>	<b>0.864</b>	<b>0.893</b>	<b>0.820</b>
Naïve	0.718	0.770	0.762	0.750	0.655	0.758	0.713	0.709

Real-valued synthetic data was created using the following process. Given  $o$  output variables, a data set is generated by selecting  $c$  points in the input space as centroids. These points are each randomly assigned a number of probability vectors. A probability vector contains a probability distribution over possible output vectors. To generate an instance, a centroid is selected at random, the input values for that instance are generated by randomly perturbing the centroid according to a Gaussian distribution. An output vector is chosen by randomly selecting an output vector according to the probability distribution of a randomly chosen probability vector for that centroid. This generation process attempts to model the fact that, for MOD problems, a portion of the input space can belong to more than one output vector. Nominal synthetic data was created using the process outlined above with one difference. To generate a centroid, a center point for each feature was chosen from  $\{0, 1, 2, 3\}$ . New inputs were generated by adding a randomly selected value from  $\{-1, 0, +1\}$  to the center point for that feature. Values above 3 were set to 3 and values below 0 were set to 0. The parameters were set to  $o \in \{2, 3, 4\}$  (with 4 possible values for each output) and  $c \in \{2, 4, 6, 8\}$ . The number of inputs was set to 3 times the number of outputs. The number of probability vectors was the same as the number of centroids, 1.5 times the number of centroids, or 2 times the number of centroids. 5000 instances were generated for each data set. This results in 12 data sets for each of 2 outputs, 3 outputs, and 4 outputs, giving a total of 36 data sets used.

The results of comparing HMONN to the naïve model for real-valued and nominal features are given in Table 1. HMONN outperformed the naïve model for the real-valued synthetic data, and the improvement was always statistically significant. This is likely due to the fact that HMONN exploits the information contained in the local neighborhood in order to produce outputs. HMONN will have more information available with more outputs. This will make the neighborhood more specific, thus giving the algorithm a higher chance of finding a correct output. HMONN outperformed the naïve model for the nominal synthetic data as well, and the improvement was always statistically significant.

## 4.2 UCI Data

The UCI repository [3] does not contain any data sets that are MOD decision problems. Therefore, MOD data sets were created from the original UCI Data sets by allowing each nominal feature to act as an output class for a derivative

**Table 2.** Results for the UCI experiments. The H columns signify the accuracy for HMONN and the NI columns signify the accuracy for the naïve independence model. Bold indicates that the model had significantly greater accuracy. The *# Significant* line indicates how many of the entries in each column were statistically significant. The *info* provides information about the UCI data sets including the number of features (Feat) and nominal features (N), and the number of derived 2, 3, and 4-output data sets.

	2-Output		3-Output		4-Output		Total		Info				
Data set	H	NI	H	NI	H	NI	H	NI	Feat	N	2-O	3-O	4-O
adult	0.255	0.268	0.136	0.139	0.075	<b>0.083</b>	0.109	<b>0.116</b>	14	8	8	28	56
anneal	<b>0.756</b>	0.511	<b>0.700</b>	0.364	<b>0.659</b>	0.254	<b>0.664</b>	0.265	38	32	32	496	4960
autos	<b>0.265</b>	0.192	<b>0.208</b>	0.118	<b>0.132</b>	0.068	<b>0.159</b>	0.088	25	10	10	45	120
car	<b>0.233</b>	0.229	<b>0.066</b>	0.063	<b>0.018</b>	0.017	0.067	0.065	6	6	6	15	20
chess	<b>0.100</b>	0.090	<b>0.020</b>	0.017	<b>0.004</b>	0.003	<b>0.024</b>	0.021	6	6	6	15	20
cmc	<b>0.358</b>	0.281	<b>0.243</b>	0.203	<b>0.173</b>	0.132	<b>0.217</b>	0.172	9	7	7	21	35
colic	<b>0.330</b>	0.255	<b>0.174</b>	0.106	<b>0.106</b>	0.048	<b>0.124</b>	0.064	22	15	15	105	455
credit-a	0.424	0.434	0.278	<b>0.297</b>	0.178	<b>0.202</b>	0.223	<b>0.245</b>	15	9	9	36	84
crx	<b>0.441</b>	0.438	0.265	<b>0.290</b>	0.161	<b>0.191</b>	0.210	<b>0.236</b>	15	9	9	36	84
heart-h	<b>0.509</b>	0.279	<b>0.401</b>	0.152	<b>0.300</b>	0.067	<b>0.357</b>	0.119	13	7	7	21	35
hepatitis	0.591	0.605	0.443	0.431	<b>0.381</b>	0.341	<b>0.401</b>	0.369	19	13	13	78	286
mush	<b>0.795</b>	0.775	<b>0.630</b>	0.590	<b>0.497</b>	0.451	<b>0.518</b>	0.473	22	22	22	231	1540
nursery	<b>0.282</b>	0.278	<b>0.091</b>	0.089	<b>0.028</b>	0.027	<b>0.069</b>	0.067	8	8	8	28	56
poker	<b>0.095</b>	0.091	<b>0.016</b>	0.015	<b>0.003</b>	0.002	<b>0.011</b>	0.011	10	10	10	45	120
post-op	<b>0.360</b>	0.346	<b>0.248</b>	0.238	0.128	0.136	0.194	0.194	8	7	7	21	35
SPECT	<b>0.661</b>	0.651	<b>0.513</b>	0.494	0.397	0.372	0.415	0.391	22	22	22	231	1540
TA	0.217	0.183	<b>0.139</b>	0.061	0.088	0.017	<b>0.146</b>	0.083	5	4	4	6	4
tic-tac	<b>0.453</b>	0.425	0.190	0.187	0.065	<b>0.074</b>	0.127	<b>0.130</b>	9	9	9	36	84
vote	<b>0.729</b>	0.708	<b>0.564</b>	0.540	<b>0.442</b>	0.417	<b>0.470</b>	0.445	16	16	16	120	560
zoo	<b>0.816</b>	0.568	<b>0.726</b>	0.492	<b>0.654</b>	0.400	<b>0.670</b>	0.420	16	16	16	120	560
# Sig	16	0	15	2	13	4	13	4					

data set. If, for example, the number of outputs was set to two, each data set would create  $n$  derived data sets where  $n$  is the number of nominal features for the chosen data set. Each of these derived data sets consists of a nominal feature combined with the original output class acting as the output classes, with all of the other features acting as inputs. Similarly, for three or four outputs the original output class is combined with two or three (respectively) nominal features to act as the outputs. The number of data sets scales linearly in the number of inputs with two outputs, quadratically with three outputs, and cubically with four outputs. This is a contrived solution, but we assume that there is some dependency between input variables and the output variable — especially for data sets from the UCI repository. Twenty UCI data sets were used for the experiments. These data sets were chosen arbitrarily from those that had more than five nominal input features. Nominal input features were necessary in order to create the derivative data sets. Information for each data set is provided in Table 2. Missing values were replaced by the mean/mode.



**Table 3.** The table showing the results of the real-world business data experiments.

	HMONN	Naïve		HMONN	Naïve
2-Output	0.508	0.465	3-Output	0.346	0.307

Table 2 shows the results for the UCI data set experiments. The table contains values for both HMONN and the naïve model compared by number of outputs. Each number is obtained by averaging the results across all the derived data sets from the original UCI data set for the given algorithm. Statistically significant results are highlighted. HMONN outperformed the naïve model 79% of the time (with 68% of the time being statistically significant, see the Total columns). In some cases there was not a significant difference. In four cases, the naïve model outperformed HMONN. HMONN outperforms the naïve model in the majority of cases. Occasionally, the naïve model performs better, but never with the same magnitude. This further demonstrates the potential of HMONN as a model to solve MOD decision problems. This also validates the assumption that there is some dependence between the output variable and the input variables in the UCI data sets.

### 4.3 Business Application Data

The motivation for defining MOD problems stems from a local business, InsideSales.com, that provided data for a real world MOD task. Due to the proprietary nature of this business data, we are only permitted to reproduce a de-identified version of this data. This data includes a two output data set and a three output data set. The data sets have fourteen nominal features and eight real-valued features. The two output data set has 32544 instances, and the three output data set has 32774 instances.

The task is to determine the timing and method to contact business leads. Business practices would imply that these variables are dependent (given the input  $\mathbf{x}$ ), the time you contact a lead depends on the method used, and the method used depends on the timing. The results are shown in Table 3. HMONN outperformed the naïve model in both cases. This shows that the improvement of HMONN seen in the UCI and synthetic data can also be seen in real-world MOD problems. The synthetic data and the real-world business data are definitely MOD problems. However the synthetic data is not necessarily representative of real data, and there is little real data. The UCI data is used to supplement the other data sources, although it can only be assumed to represent MOD data.

## 5 Conclusions

We provided a definition for MOD problems, as a well as a method to solve such problems. We have defined the Hierarchical Multi-Output Nearest Neighbor model, with a naïve independence model as a first layer and a modified nearest

neighbor model as the second layer. This model is based on the assumption that local context is a key element to solving MOD problems. HMONN consistently outperforms the baseline model, typically with statistical significance. This holds true for synthetic data, UCI repository data, and for one real-world business task.

Future work will develop solutions using other types of models (such as relaxation networks), an improved method for calculating accuracy on MOD problems, improved methods for validating new MOD algorithms, and new methods for identifying and collecting MOD data. With MOD problems, it is difficult to know how much dependency any given problem may have. Many of the data sets that we used for validation could only be assumed to have some level of dependency. A method to identify the degree of output dependency in a given data set is another piece of future work.

## References

1. Bakır, G., Hofmann, T., Schölkopf, B.: Predicting structured data. The MIT Press (2007)
2. Džeroski, S.: Multi-relational data mining: an introduction. ACM SIGKDD Explorations Newsletter 5(1), 1–16 (2003)
3. Frank, A., Asuncion, A.: UCI machine learning repository (2010), <http://archive.ics.uci.edu/ml>
4. Getoor, L., Mihalkova, L.: Learning statistical models from relational data. In: Proceedings of the 2011 international conference on Management of data. pp. 1195–1198. ACM (2011)
5. Godbole, S., Sarawagi, S.: Discriminative methods for multi-labeled classification. In: Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 22–30. Springer (2004)
6. Heath, D., Zitzelberger, A., Giraud-Carrier, C.: A Multiple Domain Comparison of Multi-label Classification Methods. Working Notes of the 2nd International Workshop on Learning from Multi-Label Data p. 21 (2010)
7. Hopfield, J., Tank, D.: Neural computation of decisions in optimization problems. Biological cybernetics 52(3), 141–152 (1985)
8. Neville, J., Rattigan, M., Jensen, D.: Statistical relational learning: Four claims and a survey. In: Workshop SRL, Int. Joint. Conf. on AI (2003)
9. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases: Part II. pp. 254–269. Springer-Verlag (2009)
10. Read, J., Pfahringer, B., Holmes, G., Frank, E.: Classifier chains for multi-label classification. Machine Learning 85(3), 333–359 (2011)
11. Taskar, B., Guestrin, C., Koller, D.: Max-margin markov networks. In: Thrun, S., Saul, L., Schölkopf, B. (eds.) Advances in Neural Information Processing Systems 16. MIT Press (2004)
12. Tsoumakas, G., Katakis, I., Vlahavas, I.: Mining multi-label data. Data Mining and Knowledge Discovery Handbook pp. 667–685 (2010)